Java Streams

(☆ A pipeline that processes a sequence of elements from a data source using a chain of intermediate / terminal operations, enabling functional-style and efficient data manipulation.)

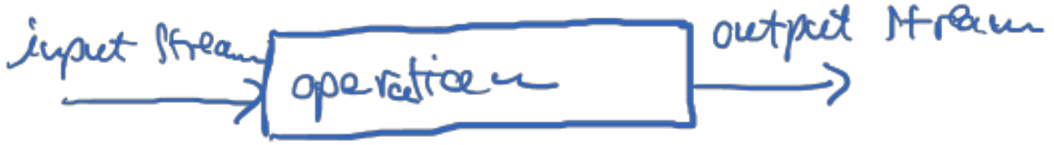
What is required for a Stream?

1. Data source

→ Stream<T>

2. Chain of intermediate operations

   intermediate operations are lazy



3. One terminal operation

   terminal operations are eager

Data Sources: Examples

$\underline{\text{1 get method}}$

$\downarrow$

java . util . Stream . Stream . generate(Supplier<T>)

.... . Stream.of(T... items)

.... . Stream.concat($\underline{\text{Stream 1}}$, $\underline{\text{Stream 2}}$)

.... . Stream . empty()

java . nio . file . Files . lines (path)

Intermediate Operations: Examples

- filter(Predicate<T>) one boolean method

  1, 2, 3, 4 → .filter((item) → item % 2 == 0) → 2, 4

- map(Function<T, R>) T → R

  "CS400", "JAVA" → .map((item) → item.toLowerCase())

  → "cs400", "java"

- limit(int) # of elements to pass through
- skip (int)

Terminal Operations: Examples

- findFirst( )
- forEach($\underline{\text{Consumer<T>}}$) ← one void method
- count( )
- min(Comparator< $\underline{\text{T}}$ >)
- max(Comparator<T>)